Towards formal methods for mathematical modeling

Ursula Martin
SRI International, Menlo Park CA
University of St Andrews, Scotland
um@dcs.st-and.ac.uk

Abstract

We survey mathematical modeling, the mathematical and computational technologies upon which it relies, and the potential sources of error. We assess formal methods and computational logic in this light, suggesting that certain well worn paths may have little to offer. We identify as opportunities for the future: analyzing requirements, assumptions and proof obligations for the assessment and confirmation of models, extending such techniques to architectures for heterogeneous distributed models with legacy components, using computational logic to extend the capabilities of computer algebra systems, and techniques for symbolic analysis.

1 Introduction

The purpose of this paper is to assess formal methods and computational logic from the point of view of mathematical modeling. It forms part of a larger research program assessing formal methods and computational logic for mathematics and its applications.

The techniques of mathematical modeling, that is of regarding a physical phenomenon as a dynamical system for the purposes of understanding and prediction, arose in the physical sciences during the twentieth century, were used widely in meteorological and defense applications and later spread to environmental, biological and geological modeling. They were transformed by modern computation, and by increasing reliance on modeling in many aspects of public policy, and have also become the keystone of US undergraduate math curriculum reform [38]. This paper concentrates on the issues arising in bioscience and environmental science, rather than on physical sciences, engineering or control theory: in particular we are considering computational rather than physical models.

In the first part of the paper we survey mathematical modeling, the math and software that it relies upon, and possible sources of error and user concern. We go into some detail, on the grounds that assessing how formal methods might be used in practice requires a general understanding of what the practice of modeling is. In the second part we consider how formal methods and computational logic might address these concerns, and identify some possible new directions.

Section 2 is a methodological aside. Section 3 contains an account of mathematical modeling, which we encapsulate as a "purposeful representation of reality". A modeler devises a "model world" to investigate some "purpose" in the "real world". A mathematical "model" of the model world is constructed using dynamical systems, and the modeler reasons within it. Almost universally today the reasoning is done with the aid of numeric or symbolic computation, so an "implementation" of the mathematical model is built in a computer system: from the "implementation" conclusions are drawn about the "model" or the "model world" and assessed against the hypotheses of the "model world" or against observations of the "real world". We may view this as a pipeline: $\{$*reality+purposes*$\} \to$ *model world* $\to$ *model* $\to$ *implementation.*

Thus modeling relies on two underlying technologies: the mathematical theories of differential equations and dynamical systems, and the computational tools of numeric or symbolic computation.

In Section 4 we give a brief account of the first of these, the mathematical theories. We describe the kind of reasoning that is typically done, and assess the correctness issues. We note in particular that the mathematician developing the theories, the toolsmith using them to devise algorithms and the modeler using those algorithms may have somewhat different perspectives.

In Section 5 we consider the second technology, and describe numeric and symbolic computation and some of the correctness concerns that arise. Numerical systems are widely used because they always give an answer: it is suggested that general software engineering issues rather than bugs in algorithms or floating point arithmetic are the main cause of error. Symbolic computation systems are much less flexible, and further problems arise because of fundamental design issues which mean that continuous math is sometimes handled incorrectly.

Sections 4 and 5 considered the underlying technologies: in Section 6 we return to the modeling process itself and assess correctness concerns. While these can arise anywhere in the pipeline, it is the assessment of a "model" or "model world", against competitors and against purposes that attracts most attention in the modeling community, and in matters such as environmental prediction (for example, querying assumptions about ground water penetration) they can be subject to heated debate. In large or legacy models even tracking built-in assumptions can be hard.

Section 7 addresses how computational logic and formal methods may address some of the correctness concerns raised in the previous sections. The correctness of the mathematical and computational technologies can in principle be addressed using techniques of computational logic: we indicate the main notions for both. In particular we report briefly on our own work using heavy duty theorem proving in PVS to provide convenient embedded reasoning tools for computational mathematics systems. However we assert that in general the modeling community are users rather than creators of mathematics and software, and are not particularly concerned to have formal developments of either the underlying material or its applications in modeling, or to replace them with new foundational approaches: these are all regarded loosely speaking as "solved problems". While in principle techniques based on improved forms of symbolic computation, or on computational logic, would allow richer reasoning about models, it is hard to see them matching the flexibility of numerical systems or overcoming the investment in existing techniques.

Correctness concerns about the modeling pipeline involve, in so far as they can be formalized, tracking of requirements and assumptions, and here we judge there to be much greater potential for formal methods from the user's point of view. We report briefly on our own experience with light formal methods for tracking requirements, assumptions and proof obligations in computational mathematics systems.

In the light of the above Section 8 sets out four main opportunities for the future: analyzing requirements, assumptions and proof obligations for the assessment and confirmation of models, extending such techniques to architectures for heterogeneous distributed models with legacy components, using computational logic to extend the capabilities of computer algebra systems and improved techniques for symbolic analysis.

2 A methodological note

It would be easy enough to tell a rosy story within the contemporary rhetoric of formal methods and computational logic of their potential for mathematical modeling, illustrated with anecdotes of unreliable predictions from unsound models or bugs in numerical code. We might then, with some effort, treat a simple differential equation or verify a numerical algorithm within our formalism of choice, argue with the aid of a large bibliography about how such methods are "growing in importance", "vital for safety critical applications of mathematical modeling", "essential for mathematicians in developing trusted proofs" and so forth, and conclude with an exhortation to the academic and commercial modeling community to take up our ideas forthwith.

We have attempted a somewhat different approach here, by identifying, albeit informally, the practice and concerns of the modeling community and how formal methods techniques might address them.

The identification of "practice" in a discipline involves finding out what people actually do, rather than what they say they do, or what others think they should do. Thus for example in [25] we showed that practice in pure mathematics research does not, as an outsider might suppose, consist in rigorous formal development but rather in the development of "good enough" proofs: this explains why computational logic engines are hardly used by pure mathematicians.

For sociologists such as Latour [22] identifying practice involves detailed observations over many months in laboratories, and careful enquiry as to whether there is any such thing as a universal or context-independent notion of scientific method, rather than "particular courses of action with materials to hand" [24].

For the purposes of this paper we gained an overview from textbooks, university courses, meetings, seminars, newsgroups, bug-reports and discussions with reflective practitioners, who included both developers and users of such software[1]. I am not aware of any thorough study into correctness concerns for modeling and what causes errors, although Mackenzie has touched on such matters in his sociological account of the development of nuclear weaponry [24]. Certainly the matter has not received the attention given to safety-critical systems. This paper can only be regarded as a pilot investigation: I conclude that, while certain individual incidents have been noted and studied, in general correctness is taken for granted, and where it is discussed it is the correspondence of models to reality, rather than the correctness of the underlying mathematics or software, that causes concern.

3 What is a model?

What is a model? A mathematical representation of reality? What is reality? What is a mathematical representation of it? Is it "out there" or "purely formal", or constructed in the minds of scientists with all kinds of motives and purposes, including the quest for truth (whatever that might be)? Questions of this kind have occupied philosophers of science for centuries. For this paper we adopt a work-a-day definition based on the standard student text of Mooney and Swift [28]: a mathematical model is a purposeful representation of reality using the tools and substance of mathematics, including computation.

A classic example is the predator-prey model whose purpose is to understand the long-term behavior of populations of predators (for example lynx) and prey (for example hares) which manifest cyclical behavior: as lynx numbers x rise more hares are eaten, so hare numbers y drop, so lynx numbers drop, so more hares survive, so lynx have more to eat, so lynx numbers increase, and so on. This is modeled by two differential equations, where $\alpha, \beta, \gamma, \delta$ represent parameters which will vary for different populations.

$$\begin{pmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \end{pmatrix} = \begin{pmatrix} \alpha & -\beta x \\ \gamma y & -\delta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \qquad (1)$$

We call this Model PP1. From these equations we may prove that $y^{\alpha} e^{-\beta y} x^{\gamma} e^{-\delta y} = K$ and hence deduce that in the model x and y do indeed manifest cyclical behavior over time for certain values of the parameters. Even without this analytic formula we can implement the equations numerically and hence draw graphs of x, y and t to display the cyclical behavior.

A simple account of modeling considers "the real world" (including hare and lynx), a "purpose" (understanding population change in hare and lynx), the "model world" consisting of assumptions we have made or chosen about the real world (for example that lynx die when there are no hares to eat), the mathematical "model" we have built of our model world using dynamical systems,[2] and the "implementation" of that model in a computer system. From the "model" or its "implementation" we can draw conclusions about the "model world" which we can then assess against the hypotheses of the model or against experimental or other understanding of "the real world". We may view this as a pipeline: $\{$*reality+purposes*$\} \rightarrow$ *model world* $\rightarrow$ *model* $\rightarrow$ *implementation.*

The predator-prey model PP1 above is an abstraction, whose purpose is to investigate the apparent cyclical nature of such populations. It tells us that if the hypotheses in the model world about the behavior of hare and lynx are satisfied, and if $\alpha, \beta, \gamma, \delta$ take certain values, then certain consequences ensue in the model, and hence by implication in the "model world". We may then use domain knowledge to give an interpretation of our conclusions for "the real world".

If we wanted to study a particular population of hares and lynx this model would not be of much use. We would need a different "model world" and a more complicated "model", which we denote by PP2. We would take other phenomena into account, for example what hares eat, and consider data, either real or simulated, on weather patterns or grass growth for our particular population. We would probably no longer have an analytical solution, and would have to rely on an "implementation" to obtain numerical, graphical or visual estimates for long term behavior. These estimates would still be contingent upon our assumptions, and the nature and quality of the data we used. PP2 might not manifest cyclical behavior at all: it might not include the equations of PP1. The mathematical relationship between our two models might be complex: it would be unlikely that, in formal method terms, one was a simple refinement of the other for instance. The distinction between these two kinds of model, roughly speaking the first more concerned with abstract principles or putative laws of nature,

[1] See acknowledgements section for more details

[2] For the purposes of this paper we ignore stochastic and discrete aspects

the second with simulations and predictions of phenomena, has sometimes been drawn by calling the former "models" and the latter "simulations". However there is no hard and fast distinction.

Both PP1 and PP2 are, in modeling terms, fairly small and straightforward, in contrast to global models of climate or population, refined over many years with complex data sets.

Once we have a model, or several models, we may investigate their solution and other properties, either mathematically or through an implementation. Models are assessed and evaluated against their purposes, or against other models that address the same or related purposes. Of particular concern is the definition and assessment of correctness.

4 Mathematical techniques

The theory

In this section we give a summary of some of the theory of differential equations and dynamical systems from the point of view of mathematical modeling applications.

What do we mean by a differential equation, and a solution? At an elementary level in a modeling text such as Mooney and Swift [28] the notion is often given only by example: for instance suppose we wish to model the motion of a particle in terms of the time and distance from an initial point (y) and the acceleration ($y'' = d^2y/dt^2$). The equation

$$y''(t) + y(t) = 0 \tag{2}$$

describes the motion at time t, any solution has the form $\phi(t) = Asin(t) + Bcos(t)$ where A and B are arbitrary constants, and a solution satisfying the initial conditions $y(0) = 1, y'(0) = 2$ is given by $\phi(t) = 2sin(t) + cos(t)$. A solution satisfying the initial conditions can be evaluated at any value of t, so that for our solution ϕ at time $t = \pi/2$ the position will be given by $\phi(\pi/2) = 2$. This equation has an explicit mathematical solution (we call this an analytic solution), but for many equations we may know only of the existence of such solutions, and numerical solutions at particular points (subject to the accuracy constraints of numerical analysis) may be all that are available to us.

"Solving" an equation involving an unknown function y and its derivatives, and conditions on the value of y at certain points, involves finding a particular (some possible such y) or a general (all possible such y) analytic solution in terms of known functions. In texts at the level of [28] various standard "cook-book" techniques are given, accompanied by reassurance and motivation for the reader. There is also particular stress on determining the qualitative or limiting behavior of the solution: does it decay over time for example.

Thus for example [28] contains the following recipe for solving first order linear differential equations of the form

$$\frac{dy}{dx} + a(x)y = b(x) : \tag{3}$$

the general solution is (sic, including sloppy variable naming)

$$y(x) = \frac{1}{\mu(x)}\left(\int \mu(x)b(x)dx + C\right) \tag{4}$$

where $\mu(x) = \exp(\int a(x)dx)$. This description elides many issues concerned with exactly when functions are defined or differentiable, or solutions exist. The standard approach of an undergraduate course in differential equations makes matters more precise: *Suppose that a and b are continuous functions on an interval I. Let $A(x)$ be a function such that $dA/dx = a(x)$. If C is any constant then the function ϕ given by*

$$\phi(x) = \exp(-A(x))\left(\int_{x_0}^{x} \exp(A(t))b(t)dt + C\right) \tag{5}$$

where x_0 is in I, is a solution of (3), and every solution has this form.

The standard treatment continues by considering existence proofs for solutions. A particularly important class is that of linear systems, of the form

$$L(y) = y^{(n)} + a_1(x)y^{(n-1)} + \ldots + a_n(x)y = b(y), \tag{6}$$

where under suitable conditions solutions always exist, though they may not have a simple closed form representation.

In the case when all the a_i are constant the solutions to $L(y) = 0$ are found by computing the eigenvalues, or roots of the characteristic equation

$$\lambda^n + a_1\lambda^{n-1} + \ldots + a_n = 0. \tag{7}$$

Thus for example when $n = 2$ the equation

$$L(y) = y'' + 2by' + cy = 0 \tag{8}$$

has general solution given by

$$\phi(x) = \begin{array}{rr} \exp(-bx)(A + Bx), & \gamma = 0 \\ \exp(-bx)(A\exp(\sqrt{\gamma}x) + B\exp(-\sqrt{\gamma}x)), & \gamma > 0 \\ \exp(-bx)(Acos(x\sqrt{-\gamma}) + Bsin(x\sqrt{-\gamma})), & \gamma < 0 \end{array} \tag{9}$$

where $\gamma = b^2 - c$. This description of the solution may be further refined to include its qualitative behavior: for example in case $\gamma = 0$, the system oscillates, and if $b > 0$ it tends to zero (is damped), if $b < 0$, it tends to infinity and if $b = 0$ it is stable.

Current mathematical research emphasizes dynamical systems, that is, roughly speaking, solution spaces of systems of differential equations like PP1. Linear systems in n variables can be expressed as a vector equation $\mathbf{X}' = \mathbf{AX}$, where $\mathbf{A}$ is an $n \times n$ matrix, and the solutions are given in terms of eigenvalues of $\mathbf{A}$. This again allows us to predict the limiting behavior of such a system, and to identify fixed points (equilibrium points) where $\mathbf{X}' = \mathbf{0}$, and behavior near to them: for example does a point near the equilibrium point move towards it (a sink) or away from it (a source). In two dimensions an analysis like (9), called a phase plane analysis, is possible: in dimensions above two chaotic phenomena can occur.

For non-linear systems like the predator-prey model there are extensive theories of existence and uniqueness of solutions. An important practical technique for investigating qualitative behavior near a fixed point is that of taking a linear approximation there and using this to do a phase plane analysis. The full mathematical analysis of such behavior, and of the underlying dynamical systems, possible chaotic behavior and so forth, requires the full apparatus of modern differential geometry.

Applications

In the initial stages the modeler may want to manipulate and transform the model and get a few rough assessments of its behavior. The next stage would be a more detailed investigation, to compare it with alternatives, to calibrate it against data, theory or other models, and to assess its performance. At a more mature stage models may be used for prediction or for reference points against other models, as components in larger systems, or refined as new data or theoretical understanding becomes available.

For example Hammersley's [12] maxims for manipulators at an early stage include: "clean up the notation, choose suitable units, reduce the number of variables, and avoid rigor like the plague as it only leads to rigour mortis", to which one would probably add today "visualize the solution".

A typical more detailed investigation might include:

- *solving a system of differential equations subject to initial values or boundary conditions:* either analytically or numerically
- *reachability analysis:* determining if there is an analytic or numeric solution satisfying a set of constraints, typically that it starts in one region and passes through another. Thus in example (2) the point $(\pi/2, 2)$ is reachable from $(0, 1)$, but $(r, 3)$ is unreachable for any value of r
- *identification of behavior near a stationary point:* for example by a phase plane analysis
- *limiting behavior over time:* for example by an eigenvalue analysis generalizing (9)
- *perturbation analysis:* to identify behaviors of the model under local variations
- *behavior as some parameter varies:* for example changes in the phase plane as a coefficient varies

Taking a formal methods perspective one might expect to see more general reasoning about properties of the solution, for example using temporal logic. Recent work in the hybrid systems community addresses this for control systems using tools such as HyTech [17], and Dutertre [7] gives examples of reasoning about upper bounds in the requirements of an avionics application, but such work does not seem to be considered at all mainstream in the modeling community. For example searches in Citeseer [3] turn up little of relevance.

Correctness issues

In analyzing correctness issues for modeling we first turn to the correctness of the underlying mathematics.

We note first that applications of modeling are not in practice a particularly rich source of novel mathematics. There is in general [28] little enthusiasm for spending a long time developing new equations for a particular modeling problem. Standard techniques, like linearisation or power-series approximation, for replacing one equation with another that behaves in roughly the same way, may be sufficient when experimenting with a number of models at an early stage. The community tends to work with a smaller number of systems which are reasonably well understood or mathematically well-behaved and which experience or consensus deems sufficient for the domain at hand.

The researcher in dynamical systems, the applied mathematician or numerical analyst 'toolsmith' and the modeler applying those techniques are doing different things. The researcher is concerned with general theories about the existence of solutions or the behavior of families of systems. The toolsmith is developing effective techniques for solving problems like those above, with the researcher's work to assure correctness. Modelers usually want to take the underlying mathematics for granted, concentrating instead on the modeling issues that arise: their mathematical interest or understanding is perhaps unlikely to go beyond a work-a-day account at the level of [28]. In particular the researcher is doing proofs in the underlying theories, the toolsmith is doing proofs about hand or machine computation techniques, and the modeler is applying those computation techniques.

We have discussed at length elsewhere [25] attitudes to correctness in the mathematical community: we identified current mathematical practice with producing conjectural mathematical knowledge by means of speculation, heuristic arguments, examples and experiments, which may then be confirmed as theorems by producing proofs in accordance with a community standard of rigour, which may be read by the community in a variety of ways. Most of the mathematics used in applications of modeling is not particularly novel, and has been subject to the usual mechanisms of community inspection through courses and text books over many years: there does not seem to be much concern from the mathematician, the toolsmith or the modeler over its correctness. As is usual in contemporary mathematical culture few are much concerned with formal proof or matters of foundation.

When a new technique arises, for example the recent growth of interest in level set methods [35], the focus of the discussion is generally on new applications, or on faster or better (for example with less instability near cusps) performance in old ones, rather than on extended discussions of correctness.

5 Computational techniques

Numerical methods

The standard, and almost universal, approach to computation for modeling, is numerical methods, which have been part of applied mathematics and the physical sciences for almost fifty years. They are widely available through standard commercial libraries such as NAG [29] and MatLab [27], and provide the basis for large software systems, usually written in FORTRAN or C and used in chemical, physical or astronomical research as well as in practical fields like engineering, meteorology and aeronautics and increasingly today in visualization and animation. Purpose-built implementations, for example, for biosciences, environmental modeling or geology are built on top of general purpose tools such as Simulink [36] which provides a graphical interface to MatLab. For example Simulink may easily be used to run the predator-prey model for different values of the parameters, generating numeric or graphical output, from which various properties of the system may be inferred.

In addition such systems can readily accommodate other inputs, for example from sensors or measuring devices, or other numerical procedures, such as curve fitting. For many problems, for example the investigation of chaotic phenomena, there are no alternative standard techniques.

From the modelers point of view the main advantage of numerical systems is that they will always give an answer, and despite the negative evidence we cite below, with sufficient user expertise are accepted as doing so sufficiently quickly and accurately, with established protocols for testing and error analysis. Numerical methods and software like NAG or Simulink are so standard and so widely used that it is hard to see them being displaced by other techniques. However the output, and properties derived from it, will always be numeric and not analytic, and support for investigating properties of the solution or parameters may be limited.

Numerical methods: correctness issues

The user of such systems can use default settings and work in ignorance of the underlying numerics, or take more detailed control using standard techniques of numerical analysis [18] to ensure results of required accuracy. Indeed, faster and more accurate numerical methods have been the main research thrust in numerical analysis over the past forty years.

A particular issue in numerical work is correctness of floating point implementation (for example the famous Pentium bug): the consistent handling of floating point arithmetic or the translation between machines with different word-lengths are recurring legacy issues. Another is convergence criteria: is the implementation robust enough to produce the same answer again for the same inputs. Kahan [21] maintains a web-site of known problems.

Yet problems persist and even expert users may be unaware of them. The author was told of a complex bug in the British Met office implementation of the multi-grid finite element method that was worth about 2% accuracy in weather forecasts. Hatton [15] reports on observations of nine independently developed large programs for seismic data processing, and shows that although the programs used the same data and were developed to the same specifications in the same language (FORTRAN), numerical disagreement grows at a rate of 1% in average absolute difference per 4000 lines of implemented code. The programs were used to analyze large scientific datasets where typically results expect around 0.001% accuracy. He concluded that in general problems were caused not by compiler or hardware errors, but by software faults, often off-by-one errors. However the matter has not received much recognition in the modeling community [16].

Symbolic computation

Symbolic computation techniques, such as those embodied in Maple or Mathematica, appear to offer a wide range of additional facilities to the modeler, especially when combined with numerical methods. Thus the *dsolve* command in Maple, or the *DSolve* command in Mathematica, can solve a wide variety of differential equations analytically, and the user can further interact with the system or write their own code, to investigate their properties. As the account of the mathematics above demonstrates, implementations rely on other symbolic computation techniques, such as integration, polynomial solving and computing eigenvalues and eigenvectors.

There is continuing lively debate over the respective merits of symbolic and numeric computation, and active research on the best way to combine the two approaches. The main drawback from the user's point of view is that computer algebra systems are simply unable to solve many of the problems listed above, either because of unsolvability or intractability. Even if there are symbolic solution techniques such systems do not scale, and there are not in general well-developed techniques for combining numeric input or techniques with symbolic ones: hence they lack the flexibility of numerical systems.

Thus for example while symbolic techniques for reachability analysis using quantifier elimination have been investigated [20], they are in general double exponential, and intractable in all but the smallest examples.

There are a few cases where symbolic techniques are better developed than numeric ones, for example the use of model checking in systems like Hytech to reason about hybrid systems, discrete combinations of control systems. There are also a few applications where symbolic systems are used in preference to numerical systems, for example in robotic or satellite motion planning.

Symbolic computation: correctness issues

By contrast with numerical techniques, users often find symbolic computation or computer algebra systems (CAS) like Maple frustrating and hard to use: see Wester [37] for a survey. Even in situations where the user is expecting them to work they may fail to produce an "obvious" answer, or produce unexpected or wrong answers, and their performance can be very unpredictable, varying widely on apparently similar inputs.

One cause of error is failure to check side-conditions: this is not so much an error as a design decision for ease of use, since even small procedures may produce large numbers of side conditions, often intractable or undecidable. This illustrates a more general design issue: there are many examples of processes (for example definite symbolic integration via the Fundamental Theorem of Calculus) where a CAS may be able to compute an answer, sometimes correct, on a large class of inputs, be provably sound on only a subclass of those inputs (where the function is continuous) and be able to check soundness easily on a smaller subclass still (for example, since continuity is undecidable, systems use a simpler check for functions with no potential poles or discontinuities). Some CAS are cautious, only giving an answer when pre-conditions are satisfied: however this means they may fail on quite simple queries. Others try and propagate the side conditions to inform the user, though this can rapidly lead to voluminous output. Mathematica and Maple generally attempt to return an answer whenever they can and leave to the user the burden of checking correctness. In [1] we have analyzed this in some detail for symbolic integration, and proposed a solution based on verified look-up tables. We extended our ideas to dynamical systems and mathematical modeling in [26], with a suite of PVS tools to check definedness and continuity, callable from Maple.

However there is a deeper reason for apparent unsoundness than failure to check for side-conditions. Formally CAS compute indefinite integrals and solve differential equations within the algebraic framework of the theory of differential fields [2]: fields with an operator satisfying $d(f.g) =$

$(df).g + f.(dg)$. When using an indefinite integral as part of an analytic calculation, for example solving a differential equation, the answers obtained algebraically may differ significantly from what is expected. For example, viewed as an element of a differential field, the derivative of $f(x) = \tan^{-1}(x) + \tan^{-1}(1/x)$ is zero, and it follows that $f(x)$ is a constant. Viewed analytically it is a step function with the value $-\pi/2$ for $x < 0$ and $\pi/2$ for $x > 0$. Thus an "unexpected" answer to a query involving $f(x)$ may be correct within the theory of differential fields, but incorrect in the usual analytic framework for differential equations we have presented above. Similarly it is easy to get Maple's *dsolve* command to display behavior which is unsound analytically, as it applies (4) without checking continuity of a and b.

This analysis should be kept in perspective however: developers of the symbolic software systems GAP [34], **axiom**[19] and Aldor [19] indicate that the majority of bug reports tend to uncover user misunderstanding, performance, or systems flaws, especially to do with portability, rather than problems with the underlying mathematics or algorithms. For example of approximately 1100 bug reports on Aldor only one reported a problem with an incorrect library implementation, involving a failure to detect a division by zero.

6 Correctness concerns for modeling

We now return to correctness concerns for the modeling process, and consider the pipeline, $\{$*reality*+ *purposes*$\}$ $\rightarrow$ *model world* $\rightarrow$ *model* $\rightarrow$ *implementation*.

One may first ask whether the "implementation" is a correct implementation of the underlying "model". In particular we may ask which aspects of its behavior are artifacts of the "implementation" (for example a poor choice of random number generator) rather than consequences of the model, or what hidden or explicit assumptions about the model have been made and how they affect the uses to which the system has been put. For example, if the system is used in a new application and predicts that $x > 3$, is this a consequence of the model, or of some implementation decision being called upon outside its domain of validity.

Heterogeneous distributed implementations often incorporate large legacy systems where the underlying assumptions may have varied over time, where later implementors may not have fully understood the original assumptions, or have incorporated variations based on new results, or where the underlying models may be incompatible. Thus for example an implementor may have hard-wired an implicit assumption about, say, the life span of a predator which is totally inaccessible to later users, and may lead to nonsensical results when combined with a different implementation.

The correctness of an implementation concerns how the "implementation" of a model matches the "model": of much greater concern in the modeling community is the assessment of the "model" against its "purposes", or against other models with the same or related purposes. In such discussions the "model" and its "implementation" may often be identified, particularly if we only have numerical information about the model. An excellent account from the point of view of environmental predictions is given in Oreskes [31].

The correctness of a "model" is in any case contingent: it says that under the hypotheses of the "model world" certain consequences occur, and the output of the implementation may be regarded as a prediction, with estimates of error being provided by mathematical analysis in the light of the model and the reliability of the data. The hypotheses of the model world may not necessarily be very clear or explicit, being part of the assumed background knowledge of domain experts. Care needs to be taken with data: for example a famous data-set on Canadian hare and lynx populations was discredited [11] when it was pointed out that the lynx and hares lived far apart and had little opportunity to eat each other. Our ability to test the correspondence of the "model world" with the "real world" depends in part on our understanding of the phenomena, and in part on the availability of sufficiently accurate data. So questions of correctness of a particular model are complicated and often subject to heated debate or compromise.

In some cases predictions may be easy to check: the occurrence of the full moon for example is readily observed and not subject to major disagreement. So if a model with a trustworthy implementation whose purpose is to predict the full moon fails to do so we may reasonably assume the "model", or the "model world" is incorrect. Even then it may not be at all clear which assumption or equation has led to the error. However most models cannot be checked in so straightforward a way: for example the average temperature of the earth needed in models of global warming is hard to measure or estimate, and in other cases it may be infeasible to check the predictions: for example safety thresholds for aircraft loads or discharge of pollutants.

Models may be known by insiders to be inaccurate, but none-the-less used as a best guess, or treated as accurate even though they are not. Mackenzie [24] reports on the debate surrounding the abandonment of nuclear weapons testing, drawing attention to the importance of tacit knowledge in the practical development of nuclear weapons, and the possibility that they might be "uninvented" if this tacit knowledge is lost. He reports scientists' claims that a computer prediction is "pretty good" if the actual yield is within 25% of prediction, and notes that during the moratorium on nuclear testing in the 1950s dependence on and confidence in computer programs increased: according to an interviewee "people start to believe the codes are actually true, to lose touch with reality.".

Experts may disagree as to the acceptability of the model: Shrader-Frechette [39] reports disagreement among two expert committees in the 1993 assessment of the proposed Yucca Mountain Waste repository site as to whether the large and well-established geological models used could reliably predict volcanic activity. We may have several competing models: for example Gilpin and Alaya [9] used experiments on competing populations of fruit-flies to test different variants of the predator-prey "model" and "model world" against the purpose of accurate prediction of fruit-fly populations. They compared their models against the accuracy of their results, favoring those where the model world made most sense biologically, and those where the model was simple and general[3]. However it may not be the case that we can always chose among competing models so readily.

Matters become more complex when we consider many-layered models, where for example testing against "the real world" may mean in practice testing against another "implementation" of a different "model" that has acquired the standing of "the real world" for practical purposes. In assessing model PP2 for example we would need numbers of hares and lynx: would we do every count by hand or use "implementations" of established "models" of wild-life numbers calibrated with key data from field studies. And how might the assumptions of the latter affect the predictions of PP2?

As we have indicated a particular concern is the combining of different models or implementations. Different models may address different parts of our purposes differently, or in choosing to model part of a larger scheme we may have to choose between several models none of which are entirely satisfactory. Assumptions may be incompatible or unclear: this is a particular issue for legacy components where assumptions may be concealed, contradictory, or have changed over time.

[3]A much argued philosophical point. It has been suggested [31] that the quest for simplicity and generality, identified with Ockham's Razor, owes more to seventeenth century theology and mathematical convenience than any evidence that simple models are better predictors than complex ones.

7 How can formal methods contribute?

Putting together the ingredients described above we may identify the business of modeling with first developing generic mathematical theories, algorithms, and implementations, both numeric and symbolic, and then modeling particular systems by implementing them within the chosen framework as part of the modeling pipeline. Correctness concerns may be raised at all levels of the process: the mathematics, the software systems, the implementations of the model and the correspondence of the model with reality. As far as we can tell this last is of most concern to the modeling community.

Formal methods, broadly construed, offers a variety of approaches.

Mathematical theories

Since the pioneering work of de Bruijn's AUTOMATH [4], developed in 1967, the theories of analysis which underlie differential equations and mathematical modeling have been developed inside various theorem provers: for recent manifestations see Dutertre's implementation of the reals inside PVS [7] or Harrison's development as far as integration in HOL [14]. As far as we are aware a full machine verification of the mathematics outlined in the previous sections has not yet been done, but it is perfectly feasible in a number of systems, using classical or constructive techniques. However while this is possible, it is hard to see how it would serve the needs of the modeling community, who regard the soundness of the underlying math as a "solved problem", established over many years in text-books and courses. They rely on mathematicians, and the usual community mechanisms of mathematics, which are remarkably averse to rigour [25], to establish correctness of the necessary mathematics: I have identified little interest in human or machine formal proof for the classical mathematics underlying the subject, the work of the toolsmith, or its routine application in modeling. This is not to write off machine checked mathematics as an endeavor, merely to say that this community sees little point to it. While logicians [8] have considered alternative axiomatizations for differential analysis I have identified no interest among the mainstream mathematical or modeling community in these matters.

Once such a development had been done it would, in principle, be possible to investigate our models directly within the prover, recasting the various queries outlined in Section 4 as proof requirements, for example the reachability results of example (2). However it is hard to see how such systems would overcome the difficulties we have already discussed for symbolic computation systems: infeasibility or intractibility mean that often there will not be an automatic proof procedure, and users will need to produce a manual proof of something whose numerical equivalent could be produced automatically. In addition any such system would need a computational component if it was to match the exploratory capacity of existing techniques, and as Section 4 shows many of the computations or proofs would require advanced symbolic computation facilities, for example to calculate eigenvalues.

Against this however we should set the advantages of abstraction, higher level proof and the handling of parameters: for example it takes laborious numerical simulation to investigate changes in the phase plane as a coefficient varies, whereas a symbolic approach merely produces a proof obligation to be discharged.

In addition, as we have argued elsewhere [25] specialized decision procedures may prove useful for some queries, for example quantifier elimination for reachability [20].

Computational techniques

As we have seen general software engineering issues have been identified as a major source of problems in both numerical and symbolic software: since these problems and formal methods approaches to them are not peculiar to modeling we do not discuss them further here. The modeling community relies on the usual mechanisms of software development, which are averse to rigour, to establish trustworthiness of its computer systems: I have identified little interest among commercial vendors in classical formal methods techniques.

It is in principle possible to implement numerical or symbolic computation inside a theorem prover, gaining reliability at a cost in performance, and both approaches have received much attention in recent years. The notorious "Pentium bug" drew attention to the unreliability of floating point implementations, and inspired Harrison's development of floating point arithmetic in HOL [13] which has had considerable commercial impact in the verification of hardware.

Such implementations of computer algebra systems have proved harder, partly because, as we have indicated, they require implementation inside a prover of specialized algorithms such as factorization. In any case, some of the unexpected behaviors of computer algebra systems arise from the algebraic representation of analysis: these would not be solved by re-implementation inside a prover. We report elsewhere [26] on an alternative approach: we built a toolkit in the PVS [32] theorem prover which automatically checks pre and side conditions such as continuity to computer algebra algorithms such as Maple's *dsolve*, thus addressing some of the difficulties caused by unsoundness in using computer algebra systems for analytic work at little extra cost to the user.

Modeling

As we have indicated the main concerns of the modeling community are with the correctness, validation and confirmation of models.

We report elsewhere [5, 6] on our lightweight formal methods approach: we built a verification condition generator in Aldor, an internal language used in developing the computer algebra systems **axiom** and Maple, and are currently developing this work in collaboration with NAG Ltd. The verification conditions are generated at compile time from user annotations, typically recording pre- and post-conditions, and can be passed to a theorem prover or used for information or documentation.

Our original motivation was particularly that of assisting the user of libraries where the code itself might be trusted, but the assumptions or preconditions for its correct use were ill-documented. We are currently experimenting with the use of these annotations for documenting requirements and assumptions in legacy models.

However there appear to be some differences between the needs here and those of design or requirements engineering: in particular there are cases where it seems useful to record assumptions or domain knowledge that does not affect the state or output of the module where it is recorded or assumed, but may be significant elsewhere. It is not entirely obvious to us how to map the modeling pipeline to frameworks such as the reference model of Gunter et al [10], which is based on domain knowledge, requirements, specifications, program and program platform.

We note that these matters are beginning to receive commercial attention: Lemma 1 Ltd [23] report on their ClawZ system which translates Simulink diagrams into Z specifications, and the UK company QSS [33] have interfaced their DOORS requirements tool to Simulink.

8 Some new directions

The previous section paints a somewhat depressing picture, suggesting that many areas which have received considerable research attention are unlikely to have much effect on the practice of modeling. We might sum up by observing that the modeling community are users rather than creators of mathematics and software, and by and large take both the mathematical theories underlying their work and the largely commercial computer systems that implement them pretty much for granted as "solved problems". The main concerns lie elsewhere and there is little interest in or motivation for change, and a heavy personal and financial investment in existing technologies.

We can none-the-less outline some ways ahead. The modeling community, like many others, are interested in new methods that fit their present world view, address their main concerns or improve or extend existing techniques or software.

The correctness, validation and confirmation of models is of primary importance to the modeling community, and of particular concern when these impact public policy in matters such as nuclear waste disposal. It is the assumptions, data and choice of model that seem to matter here, not questions about correctness of the underlying mathematics or software once the model has been chosen. We are not even aware of a suitable framework for the analysis of requirements, specifications, assumptions and proof obligations for modeling within our pipeline: an extension of the reference model of Gunter et al [10] may be appropriate. Computational logic has a useful role to play in monitoring and analysis here, and hence in reasoning directly about the assumptions of the "model world", the "model" and the "implementation".

Heterogeneous distributed models are of particular current interest, put together for example across the Internet, with disparate or legacy components where assumptions may be concealed, contradictory, or have changed over time. An engine for managing requirements and assumptions would be a key component technology of robust architectures for linking such heterogeneous models. Particular care would need to be taken over the layering issues indicated above. Projects such as Open Math [30], which attempt to provide reliable interface mechanisms for heterogeneous mathematical systems using type inference seem relevant here also.

New analytic, numerical or visualization techniques which leverage off the established mathematical and computational framework and extend its functionality are of interest. For example, as we have seen, computer algebra systems are useful in the analytic study of dynamical systems, especially those with parameters, but these are error prone: extending them using computational logic engines as we have indicated above adds functionality at little cost to the user.

Symbolic analysis Numerical analysis software does continuous mathematics numerically, computer algebra software does continuous mathematics symbolically by algebraic means, but no software yet does continuous mathematics symbolically by analytic means, and it is not clear how it should be done. As we have indicated this is the underlying reason for the deficiencies of computer algebra systems: solving it would indeed make possible a new generation of useful computational tools. It is not enough to formalize existing computer algebra systems based on differential rings: these will not give us true computational analysis. It is not enough to prove theorems about real analysis inside a theorem prover: we need to be able to do computations like those described in Section 3 as well.

We urge the formal methods and computational logic community to take up these challenges.

Acknowledgements

The author acknowledges support from the UK EPSRC under grant GR/L48256 and from NAG Ltd, with additional sabbatical support from SRI Menlo Park and the UK Royal Academy of Engineering. She thanks colleagues at Waterloo Maple, Mathworks, MathEngine and NAG Ltd, and in the Schools of Mathematics and of Biological Sciences at St Andrews, and the Departments of Mathematics and of Geology at Stanford for helpful discussions: any misrepresentations here are her own.

References

[1] A Adams, H Gottliebsen, S Linton, U Martin. VSDITLU: a verified symbolic definite integral table look-up Proc CADE 16, LNAI 1632, 112-126, Springer 1999

[2] M. Bronstein. *Symbolic integration. I.* Springer, 1997.

[3] Research Index, http://citeseer.nj.nec.com/cs

[4] N de Bruijn, The mathematical Language AUTOMATH, its usage, and some of its extensions, Symposium on Automatic Demonstration, Lecture Notes in Mathematics 125, Springer 1968

[5] Martin Dunstan, Tom Kelsey, Steve Linton and Ursula Martin Lightweight formal methods for computer algebra systems In ISSAC'98, ACM Press, 1998

[6] Martin Dunstan, Tom Kelsey, Steve Linton and Ursula Martin Formal Methods for Extensions to CAS Proc FM'99, LNCS 1709, 1758-1777, Springer 1999

[7] B. Dutertre. Elements of Mathematical Analysis in PVS. Proc TPHOLS 9, LNCS 1125, Springer 1996.

[8] S Feferman, Why a little bit goes a long way: Logical foundations of scientifically applicable mathematics, in PSA 1992, Vol. II, 442-455, 1993.

[9] M E Gilpin and F J Ayala, Global models of growth and competition, Proc Nat Acad Sci USA 70, 3590-3593, 1973

[10] Carl A Gunter et al, A reference model for requirements and specifications, to appear IEEE Software.

[11] C A S Hall, Assessment of theoretical models, Ecological Modeling 43, 5-31, 1988

[12] J Hammersley, Maxims for manipulators, Bull I M A 9 (1973) 276.

[13] J Harrison, Floating point verification in HOL. In Proc TPHOLS 8, LNCS 971, 186-199, Springer 1995

[14] J Harrison, Constructing the Real Numbers in HOL, Formal Methods in System Design 5 (1994) 35-59

[15] Leslie Hatton, The T-experiments: errors in scientific software, IEE Computational Science and Engineering, 4, 27-38, 1997

[16] Leslie Hatton, personal communication.

[17] Thomas Henzinger and Pei-Hsin Ho, HyTech: The Cornell Hybrid Technology Tool, Hybrid Systems II, LNCS 999, 265-294, Springer, 1995.

[18] N Higham, Accuracy and Stability of Numerical Algorithms, SIAM Press 1996

[19] R D Jenks and R S Sutor, **axiom**: The Scientific Computation System, Springer 1992

[20] M Jirstrand, Nonlinear Control System Design by Quantifier Elimination, Journal of Symbolic Computation, 24, 137-152, 1997.

[21] W Kahan, http://www.cs.berkeley.edu/ kahan

[22] B Latour and S Woolgar, Laboratory Life: the Social Construction of Scientific Facts, Sage, London, 1979

[23] ClawZ: Lemma 1 Ltd, http://www.lemma-one.com

[24] Donald Mackenzie, The uninvention of nuclear weapons, Chapter 10 of Knowing Machines, MIT Press 1998.

[25] U Martin Computers, reasoning and mathematical practice In Computational Logic, NATO Adv. Sci. Inst. Ser. F Comput. Systems Sci., Springer 1998

[26] U Martin and H Gottliebsen, Computational logic support for differential equations and mathematical modeling, submitted.

[27] Matlab, http://www.matlab.com

[28] D Mooney and R Swift, A course in mathematical modeling, MAA press, 1999

[29] NAG Libraries, http://www.nag.co.uk

[30] OpenMath, http://www.openmath.org

[31] N Oreskes et al, Verification, validation and confirmation of numerical models in the earth sciences, Science 263, 1994, 641–646

[32] S Owre, S Rajah, J Rushby, N Shankar. PVS: combining specification, proof checking, and model checking. In Proc CAV 8, LNCS 1102, 411-414 Springer, 1996

[33] DOORS, http://www.requirements.com

[34] Martin Schönert et al, GAP: groups, algorithms, and programming, `www-gap.st-and.ac.uk`

[35] J Sethian, Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science, Cambridge University Press, 1999

[36] Simulink, http://www.simulink.com

[37] M Wester, Computer Algebra Systems : A Practical Guide, Wiley 1999

[38] Westpoint Consortium, Interdisciplinary Lively Application Projects, MAA Press, 1997

[39] K Shrader-Frechette, Science Versus Educated Guessing: Risk Assessment, Nuclear Waste, and Public Policy, BioScience 46, 1996